Dynamic programming

12/6/2017

Fibonacci + LCS

Properties of a problem that can be solved with DP

- Simple Subproblems
  - We should be able to break the original problem to smaller subproblems that have the same structure
- Optimal Substructure of the problems
  - The solution to the problem must be a composition of subproblem solutions
- Subproblem Overlap
  - Optimal subproblems to unrelated problems can contain subproblems in common

index	0	1	2	3	4	5	6	7	...
value	0	1	1	2	3	5	8	13	...

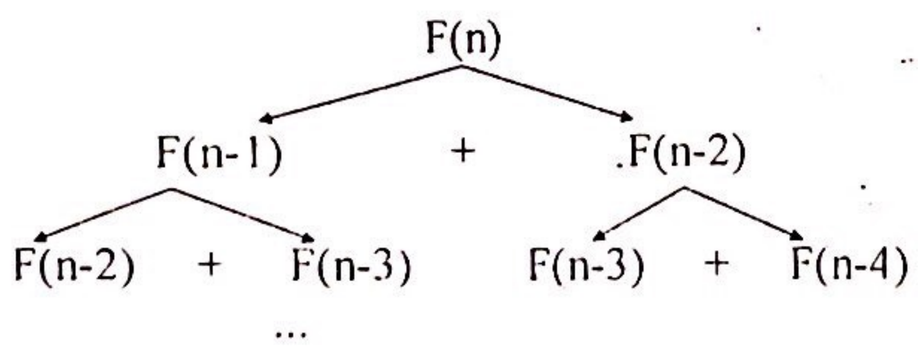
Example: Fibonacci numbers

- Recall definition of Fibonacci numbers:

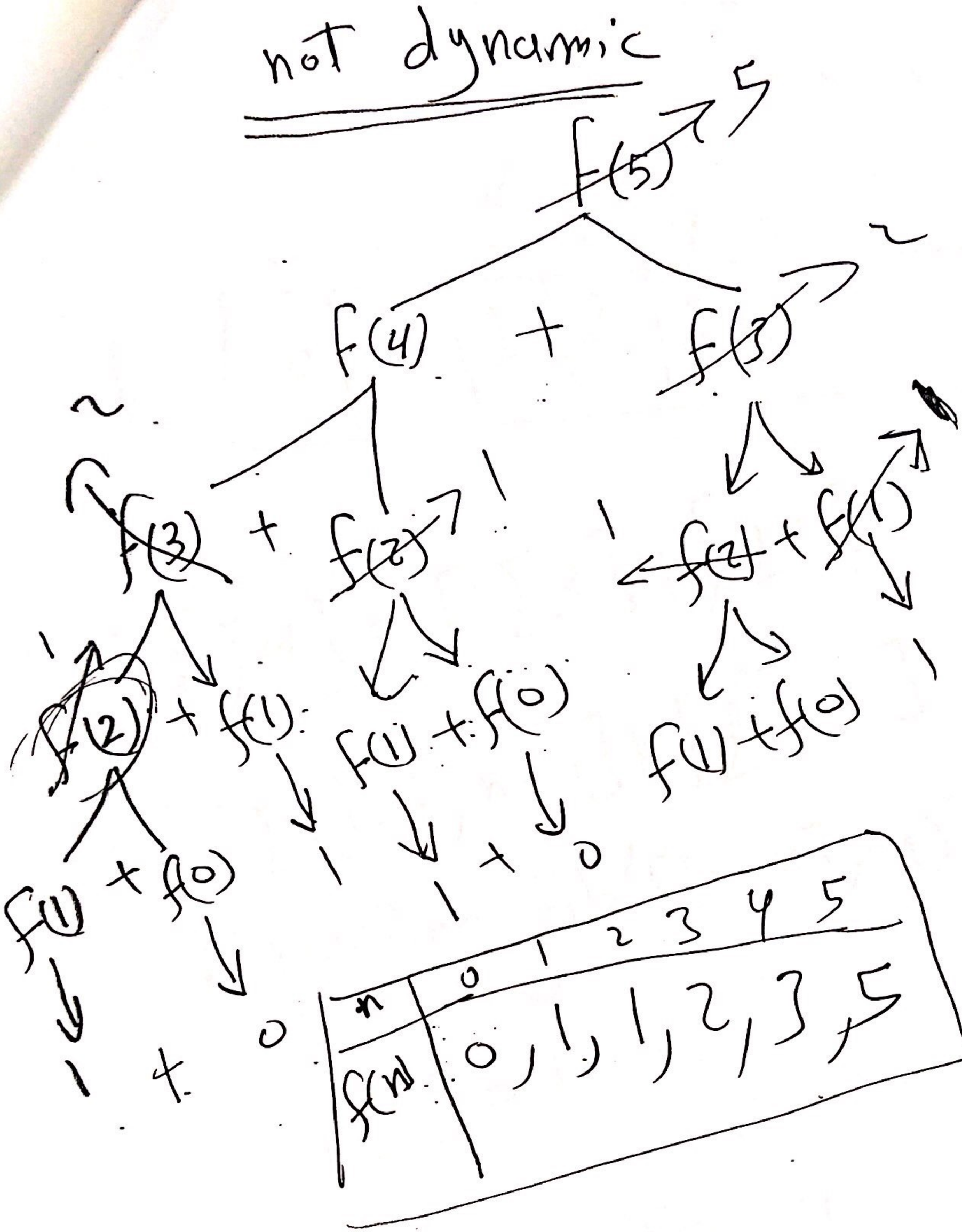
  $F(n) = F(n-1) + F(n-2)$
  $F(0) = 0$
  $F(1) = 1$

  $F(7) = F(6) + F(5)$

- Computing the nth Fibonacci number recursively (top-down):

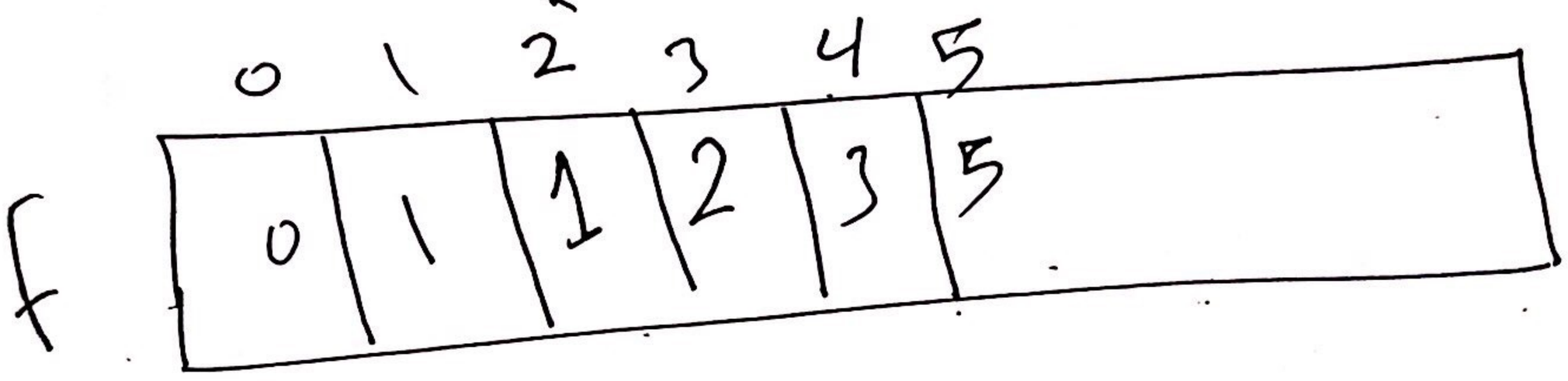


not dynamic



n	0	1	2	3	4	5
f(n)	0	1	1	2	3	5

dynamic solution



ويكون $f[4] = f[3] + f[2]$

وعلى $f[n] = f[n-1] + f[n-2]$

Algorithm fib(F[0..n], n){

$F[0] = 0$

$F[1] = 1$

for $I = 2$ To n

$F[n] = F[n-1] + F[n-2]$

return $F[n]$

}

$I = 2$
$I = 3$
$I = 4$
$I = 5$

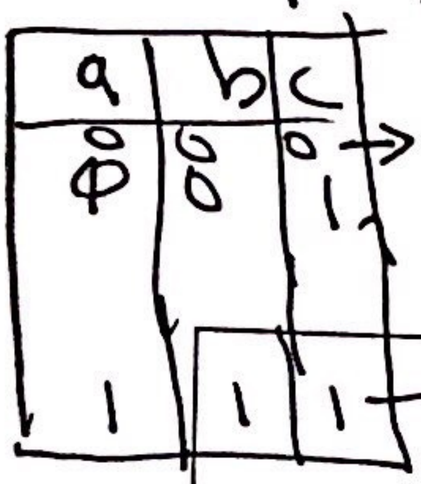
$X = \{a, b, c\}$

$P(X) = \{\phi, X, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}\}$

$|X| \quad 3$

$2 = 2 = 8$

12/6/2017



Example: Fibonacci numbers

Computing the nth Fibonacci number using bottom-up iteration and recording results:

F(0) = 0
F(1) = 1
F(2) = 1+0 = 1
...
F(n-2) =
F(n-1) =
F(n) = F(n-1) + F(n-2)

0	1	1	. . .	$F(n-2)$	$F(n-1)$	$F(n)$

Efficiency?
- time
- space

DP Example: Longest Common Subsequence (LCS)

- *Longest common subsequence* (*LCS*) problem:
  - Given two sequences x[1..m] and y[1..n], find the longest subsequence which occurs in both
  - Ex: x = {A B C B D A B }, y = {B D C A B A} (n)
  - {B C} and {A A} are both subsequences of both sequences (m)
    - *What is the LCS?*
  - Brute-force algorithm: For every subsequence of x, check if it is a subsequence of y
    - *How many subsequences of x are there?* 2^m
    - *What will be the running time of the brute-force algorithm?*

Longest Common Subsequence (LCS)

- DP algorithm: solve subproblems until we get the final solution
- Subproblem: first find the LCS of *prefixes* of X and Y.
- This problem has *optimal substructure*: LCS of two prefixes is always a part of LCS of bigger strings

Longest Common Subsequence (LCS)

Application: comparison of two DNA strings

Ex: X= {A B C B D A B }, Y= {B D C A B A}

Longest Common Subsequence:

X= A B C B D A B

Y= B D C A B A

Brute force algorithm would compare each subsequence of X with the symbols in Y

LCS Algorithm

- Brute-force algorithm: 2^m subsequences of X to check against n elements of Y: $O(n.2^m)$
- We can do better: for now, let's only worry about the problem of finding the *length* of LCS
  - When finished we will see how to backtrack from this solution back to the actual LCS

LCS Algorithm

- First we will find the length of LCS. Later we will modify the algorithm to find LCS itself.
- Define $x[i], y[j]$ to be the prefixes of X and Y of length i and j, respectively
- Define $c[i,j]$ to be the length of LCS of $x[i]$ and $y[j]$
- Then the length of LCS of X and Y will be $c[m,n]$

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- We start with $i = j = 0$ (empty substrings of x and y)
- Since $x[0]$ and $y[0]$ are empty strings, their LCS is always empty (i.e. $c[0,0] = 0$)
- LCS of empty string and any other string is empty, so for every i and j: $c[0,j] = c[i,0] = 0$

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- When we calculate $c[i,j]$, we consider two cases:
- **First case:** $x[i]=y[j]$
  - One more symbol in strings X and Y matches, so the length of LCS of $x[i]$ and $y[j]$ equals to the length of LCS of smaller strings $x[i-1]$ and $y[j-1]$, plus 1

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- **Second case:** $x[i] != y[j]$
  - Symbols don't match and the length of LCS($x[i]$, $y[j]$) is the maximum of LCS($x[i]$, $y[j\text{-}1]$) and LCS($x[i\text{-}1]$, $y[j]$)
- Why not just take the length of LCS(X_{i-1}, Y_{j-1}) ?

LCS recursive solution

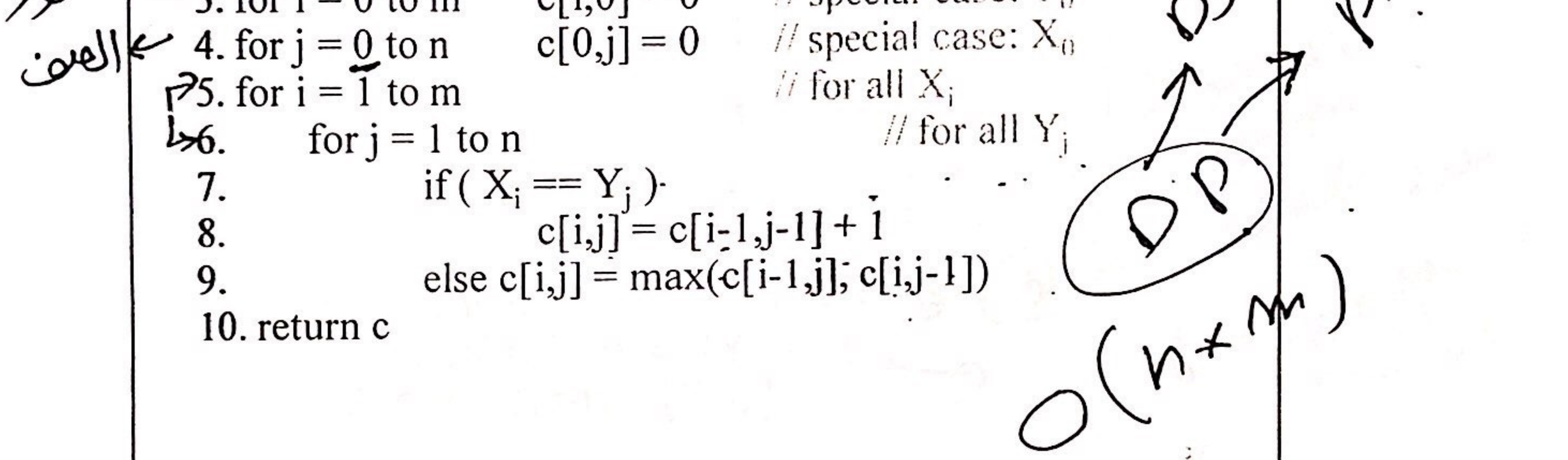
X=abc and Y=db

c[3,2]=max(c[3,1],c[2,2])=max(0,1)=1

whereas c[3,2] != c[2,1]=0

LCS Length Algorithm

```
LCS-Length(X, Y)
1. m = length(X)  // get the # of symbols in X
2. n  = length(Y) // get the # of symbols in Y
3. for i = 0 to m        c[i,0] = 0        // special case: Y_0
4. for j = 0 to n        c[0,j] = 0        // special case: X_0
5. for i = 1 to m                          // for all X_i
6.       for j = 1 to n                                // for all Y_j
7.             if ( X_i == Y_j )
8.                   c[i,j] = c[i-1,j-1] + 1
9.             else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c
```



LCS Example

We'll see how LCS algorithm works on the following example:

- X = ABCB
- Y = BDCAB
- What is the Longest Common Subsequence of X and Y?

  LCS(X, Y) = BCB

  X = A **B** **C** **B**

  Y = **B** D **C** A **B**

LCS Example (0)

X=ABCB
و BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

X = ABCB; m = |X| = 4 → 5
Y = BDCAB; n = |Y| = 5 →
Allocate array c[5,6]

BCB

LCS Example (1)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0					
2	B		0					
3	C		0					
4	B		0					

for i = 0 to m c[i,0] = 0
for j = 0 to n c[0,j] = 0

LCS Example (2)

ABCB
BDCAB

i		Xi	j=0	1 Yj = B	2 D	3 C	4 A	5 B
0			0	0	0	0	0	0
1	A		0	0				
2		B	0					
3		C	0					
4		B	0					

C

i

```
if ( Xi == Yj )
        c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max( c[i-1,j], c[i,j-1] )
```


LCS Example (3)

ABCB
BDCAB

i		Xi	j=0	1 Yj = B	2 D	3 C	4 A	5 B
0			0	0	0	0	0	0
1	A		0	0	0	0		
2		B	0					
3		C	0					
4		B	0					

```
if ( Xi == Yj )
        c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max( c[i-1,j], c[i,j-1] )
```

LCS Example (4)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	
2	B		0					
3	C		0					
4	B		0					

i = 1
j = 4

if (X_i == Y_j)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (5)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0					
3	C		0					
4	B		0					

if (X_i == Y_j)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (6)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1				
3	C		0					
4	B		0					

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (7)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	
3	C		0					
4	B		0					

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (8)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0					
4	B		0					

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (10)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1			
4	B		0					

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (11)

ABCB
BDCAB

i		j	0		1		2		3		4		5
			Yj		B		D		C		A		B
0	Xi		0		0		0		0		0		0
1	A		0		0		0		0		1		1
2	B		0		1		1		1		1		2
3	C		0		1		1		2				
4	B		0										

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (12)

ABCB
BDCAB

i		j	0		1		2		3		4		5
			Yj		B		D		C		A		B
0	Xi		0		0		0		0		0		0
1	A		0		0		0		0		1		1
2	B		0		1		1		1		1		2
3	C		0		1		1		2	→	2	→	2
4	B		0										

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (11)

ABCB
BDCAB

i		j	0		1		2		3		4		5
			Yj		B		D		C		A		B
0	Xi		0		0		0		0		0		0
1	A		0		0		0		0		1		1
2	B		0		1		1		1		1		2
3	C		0		1		1		2				
4	B		0										

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (12)

ABCB
BDCAB

i		j	0		1		2		3		4		5
			Yj		B		D		C		A		B
0	Xi		0		0		0		0		0		0
1	A		0		0		0		0		1		1
2	B		0		1		1		1		1		2
3	C		0		1		1		2	→	2	→	2
4	B		0										

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (13)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1				

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (14)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (15)

ABCB
BDCAB

i		j	0		1		2		3		4		5
			Yj		B		D		C		A		B
0	Xi		0		0		0		0		0		0
1	A		0		0		0		0		1		1
2	B		0		1		1		1		1		2
3	C		0		1		1		2		2		2
4	B		0		1		1		2		2		3

if ($X_i == Y_j$)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = **max**(c[i-1,j], c[i,j-1])

LCS Algorithm Running Time

- LCS algorithm calculates the values of each entry of the array $c[m,n]$
- So what is the running time?

O($m.n$) since each $c[i,j]$ is calculated in constant time, and there are $m.n$ elements in the array

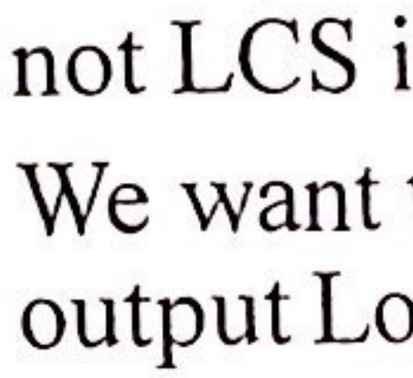
How to find actual LCS

- So far, we have just found the *length* of LCS, but not LCS itself.
- We want to modify this algorithm to make it output Longest Common Subsequence of X and Y

Each $c[i,j]$ depends on $c[i\text{-}1,j]$ and $c[i,j\text{-}1]$

or $c[i\text{-}1,j\text{-}1]$

For each $c[i,j]$ we can say how it was acquired:



For example, here

$c[i,j] = c[i\text{-}1,j\text{-}1] + 1 = 2+1=3$

How to find actual LCS

- Remember that

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- So we can start from $c[m,n]$ and go backwards
- Whenever $c[i,j] = c[i\text{-}1, j\text{-}1]+1$, remember $x[i]$ (because $x[i]$ is a part of LCS)
- When $i=0$ or $j=0$ (i.e. we reached the beginning), output remembered letters in reverse order

Finding LCS

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

Finding LCS (2)

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

LCS (reversed order): B C B

LCS (straight order): B C B

Conclusion

- Dynamic programming is a useful technique of solving certain kind of problems
- When the solution can be recursively described in terms of partial solutions, we can store these partial solutions and re-use them as necessary
- Running time (Dynamic Programming algorithm vs. naïve algorithm):
  - LCS: $O(m.n)$ vs. $O(n.2^m)$

Reading

Chapter 8

Anany Levitin, Introduction to the design and analysis of algorithms, 3rd Edition, Pearson, 2011.

LCS Length Algorithm

```
LCS-Length(X, Y)
1. m = length(X)  // get the # of symbols in X
2. n  = length(Y) // get the # of symbols in Y
3. for i = 1 to m        c[i,0] = 0      // special case: Y_0
4. for j = 1 to n        c[0,j] = 0      // special case: X_0
5. for i = 1 to m                        // for all X_i
6. for j = 1 to n                        // for all Y_j
7.       if ( X_i == Y_j )
8.             c[i,j] = c[i-1,j-1] + 1
9.       else c[i,j] = max( c[i-1,j], c[i,j-1] )
10. return c
```

LCS Example

We'll see how LCS algorithm works on the following example:

- X = ABCB
- Y = BDCAB

What is the Longest Common Subsequence of X and Y?

LCS(X, Y) = BCB

X = A **B** C **B**

Y = **B** D **C** A **B**

LCS Example (0)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			y_j →	B	D	C	A	B
	x_i ↓							
0			0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1 ←	1	1	1	2
3	C		0	1	1	2 ←	2	2
4	B		0	2	2	2	2	3

$X = ABCB;\quad m = |X| = 4$

$Y = BDCAB;\ n = |Y| = 5$

Allocate array c[5,4]

BCB

LCS Example (1)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

for i = 1 to m c[i,0] = 0
for j = 1 to n c[0,j] = 0

LCS Example (2)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0				
2	B		0					
3	C		0					
4	B		0					

$$\text{if } (X_i == Y_j)$$
$$c[i,j] = c[i-1,j-1] + 1$$
$$\text{else } c[i,j] = \max(c[i-1,j], c[i,j-1])$$

LCS Example (3)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0		
2	B		0					
3	C		0					
4	B		0					

$$\text{if } (X_i == Y_j)$$
$$c[i,j] = c[i-1,j-1] + 1$$
$$\text{else } c[i,j] = \max(c[i-1,j], c[i,j-1])$$

LCS Example (4)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	
2	B		0					
3	C		0					
4	B		0					

$$\text{if } (X_i == Y_j)$$
$$c[i,j] = c[i-1,j-1] + 1$$
$$\text{else } c[i,j] = \max(c[i-1,j], c[i,j-1])$$

LCS Example (5)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1 →	1
2	B		0					
3	C		0					
4	B		0					

$$\text{if } (X_i == Y_j)$$
$$c[i,j] = c[i-1,j-1] + 1$$
$$\text{else } c[i,j] = \max(c[i-1,j], c[i,j-1])$$